FACULTY OF SCIENCE

DEPT. OF COMPUTER SCIENCE

Locally-Aggregated Sentence Embeddings

A state-of-the-art non-parameterized method

Yuxin Wang

Supervisor: Prof. Sien Moens
Affiliation *Language Intelligence lab*

Daily Supervisor : *Ruiqi Li*
Affiliation *Language Intelligence lab*

Academic Supervisor: *Rémi Emonet*
Affiliation *Lab Hubert Curien*

Master thesis submitted in fulfillment
of the requirements for the degree in
Master of MLDM in UJM

Academic year 2018-2020

Acknowledgements

This year we are all doing something that no one has ever done before.

At the end of my master's education, I want to show sincere thanks to the Machine Learning and Data Mining team. Thank you for admitting me two years ago; Thank you for the high-quality education you gave; Thank you for providing me the opportunity to come to Leuven to continue my interests on Probabilistic Graphical Model and Natural Language Processing; Thank you for supporting my life with the scholarship. I appreciate and remember everything you did from the bottom of my heart. Moreover, I want to thank *Prof. Remi Emonet* and *Prof. Baptiste Jeudy* who gave me an opportunity to study probabilistic graphical models during the internship at Lab Hubert Curien. It triggers my interests and motivates me to step forward.

I want to thank *Prof. Sien Moens* from Language Intelligence and Information Retrieval (LIIR) research lab, KU Leuven, who accepts me as an intern and allows me to continue the research during this special period. I want to thank Ruiqi Li, my daily supervisor, keeps in touch with me, corrects my ideas, and provides advice on this paper while China, my country, is suffering from COVID-19, she is suffering from COVID-19.

Last but not the least, I want to thank everyone who stands against COVID-19. The outbreak of the virus is out of the expectation of the world. Those people, who are my peers or older or younger than me make great efforts to make our lives back on track. Without them, it is not possible to finish the paper in peace.

I received a lot of help during my master's study. I thank everyone who once showed the kindness to me. Thank you.

Abstract

This paper is written during the internship at the Language Intelligence and Information Retrieval (LIIR) research lab[1] under the supervision of Prof. Sien Moens[2] and Ruiqi Li[3]. LIIR lab is a part of the Human Computer Interaction (HCI) unit at the Department of Computer Science, KU Leuven. The members of the lab mainly focus on Natural Language Processing, Machine Learning and Probabilistic Graphical Models.

In this paper, we present a non-parameterized method to generate sentence representations (sentence embeddings) on the basis of word representations (word embeddings). No training is needed in our method. The evaluation shows that our approach can capture the semantic relationships among sentences. It achieves state-of-the-art performance on Semantic Text Similarity (STS) tasks among all non-parameterized models and is a strong competitor (the second best) to the best parameterized model trained for months. Computing one sentence with 20 words takes about $7.39 \times 10^{-4}s$, which is 31 times faster than the best model.

The idea of this paper is inspired by the self-attention mechanism introduced in the Transformer architectureVaswani et al. (2017), Vector of Locally-Aggregated Word Embeddings (VLAWE) Ionescu and Butnaru (2019) applied to generate document-level representations, and Semantic Subspace Sentence Embedding (S3E) Wang et al. (2020). Self-attention allows the Transformer to build the word/token embeddings that largely depend on *important* words from its context. VLAWE shows that the locally-aggregated word embeddings successfully in detecting the semantic relationships of documents. S3E takes the same idea and alleviates the problem of the high dimensionality in VLAWE. In this paper, we first model *Universal Sentence* using the idea of locally-aggregated word embeddings. Then relationships between sentences and *Universal Sentence* are used to build the sentence embeddings. By taking the idea of self-attention, we make the sentence embeddings largely depend on limited aforementioned relationships, which means the value distribution in a sentence embedding is skewed.

Our code is available on Github[4].

[1]https://liir.cs.kuleuven.be/

[2]sien.moens@cs.kuleuven.be

[3]ruiqi.li@kuleuven.be

[4]https://github.com/joseph-mutu/LIIR-Intern

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Natural Language Processing (NLP) is the jewel of the crown of AI whose main task is to deal with natural languages used by humans. But the main features of languages are the diversity and the flexibility. Hence, how to represent a word/sentence that can be recognized by computers has garnered a lot of attention for decades. We hope that good representations could capture fine-grained semantic and syntactic relationships of words/sentences. For example, good representations method are expected to have higher cosine similarities for words like *car* and *motorcycle* or sentences like *how old are you* and *what is your age* as they are semantically similar. But for sentences like *how old are you* and *my phone is good*, they are expected to have lower cosine similarities.

Word Embeddings

The simplest way to denote a word is called *One-hot*, which is to treat each word as an independent dimension. Hence, each word can be represented as a $\mathbb{R}^{|V| \times 1}$ vector, where $|V|$ is the size of the vocabulary. This method has obvious drawbacks:

1. **Flexible languages.** There are new words that appear on the web every day in modern society. For example, 'Awesomesauce' means extremely good, which is the combination of 'awesome' + 'sauce'. The *one-hot* representation can not deal with such new words.

2. **High dimensionalities.** Take Chinese for example, there are 47035 Chinese characters collected in the Kang-Xi dictionary. If each character is treated as one dimension, each word will be represented as a vector with 47035 dimensions.

3. **Orthogonality.** Since one-hot representation treats each word as one dimension, which means two words are orthogonal. For example, $vector(car)^T \cdot vector(motorcycle) = vector(motorcycle)^T \cdot vector(car) = 0$. It can not capture the similarity and semantic relations among words.

To overcome the drawbacks of the one-hot, several methods are introduced. According to their nature, it can be divided into two groups.

Count-based. TFIDF builds word representations from the co-occurrence matrix by considering two terms: term frequency and document frequency. GloVe Pennington et al. (2014) trains word embeddings based on a loss function that takes advantage of the co-occurrence statistics of the corpus.

Window-based. The representative model is Word2Vec proposed by Mikolov et al. (2013a). It builds a shallow neural network to construct word embeddings from the amount of unlabeled text. It probes the semantic relationships of words by using their context and succeeds in capturing the semantic differences. For example, the analogy "Kind is to Queen as man is to women" can be represented by $vector(King) - vector(Man) + vector(Woman) = vector(Queen)$ in vector space.

Sentence Embeddings

The success of word embeddings motivates the NLP community to search for methods to build representations of longer pieces of text - sentences or documents. The existing approaches can be categorized into non-parameterized methods and parameterized methods.

Non-parameterized Methods

The simplest method is called Bag-of-Words (BOW), which takes the average of word embeddings contained in a sentence/document. BOW assigns each word an equal weight in building the sentence embeddings. It is not reasonable because a sentence can contain meaningless words like *a, the, of, that, etc.* Arora et al. (2016b) proposes a weighted average approach that can beat some deep-learning models. By constructing the orthogonal basis of pre-trained word embeddings, Yang et al. (2019) achieves good performance on various tasks. Wang et al. (2020) analysis the subspace of word embeddings and use the covariance matrix to construct sentence embeddings. These methods are parameter-free, require no training, and quite fast. But non-parameterized methods are normally designed for specific usages. They are competitive with parameterized methods on specific tasks but are still inferior on other downstream tasks.

Parameterized Methods

Neural-Network Language Models (NNLM) have complex architectures and are trained from the mass of labeled or unlabeled data. Compared with non-parameterized methods, they normally need much more computational power and can be more generalized. They can be used for various tasks by adding an additional layer on top of networks. For example, the introduction of BERT Devlin et al. (2018) has set state-of-art performances on various NLP tasks.

Contribution

In this paper, we present a non-parameterized method that specially focuses on modeling the semantic differences of sentences. Moreover, by exploring the Zipf's law of locally-aggregated word embeddings, we shed light on why clustering algorithms could have an impact on the quality of sentence embeddings. we posit that clustering algorithms are doing a similar job as Self-attention introduced in the Transformer architecture Vaswani et al. (2017).

This paper has the following contributions:

1. The proposed method outperforms other non-parameterized models and can beat most parameterized models like InferSent Conneau et al. (2017), Universal Sentence Encoder

Cer et al. (2018), SBERT Reimers and Gurevych (2019) by 5 points, 2 points, and 1 point in average respectively on Semantic Text Similarity (STS) tasks.

2. The proposed method is about 1.2 times faster than InferSent, 7 times faster than SBERT, 20 times faster than Universal Sentence Encoder, and 31 times faster than the best parameterized model SBERT-WK.

3. To the best of our knowledge, this paper is the first one that investigates the properties of locally-aggregated word embeddings and their impact on sentence embeddings.

The paper is organized as follows. In Chapter 2, we discuss Related Work. In Chapter 3, mainstream word embedding techniques are reviewed. We will describe the proposed method in detail in Chapter 4. In Chapter 5, experiments with respect to the performance and the computational speed of the proposed method will be shown. Finally, the conclusion is given in Chapter 6.

Chapter 2

Related Work

2.1 Non-Parameterized Methods

A. Weighted Sum Sentence Embeddings

Averaged word embeddings (Bag of Words, BOW) is a common strategy to build sentence embeddings and is treated as a base-line. But it has obvious problems: (1) the sentence embedding can be dominated by words without actual meaning. (presumably *the, a, an, etc.*) Such words appear frequently and regardless of the context. They disturb the semantic meaning of sentence embeddings. The semantic meaning of a sentence should more rely on "important" words (presumably, *love,like, etc.*. (2) The positional information is ignored in BOW. In a scenario, say *A beats B*, positions of words matter. But BOW builds the same representation for *A beats B* and *B beats A*.

Arora et al. (2016b) proposed a weighted average method called *smooth inverse frequency (SIF)* by introducing new smoothing terms under the same assumption of the latent generative model Arora et al. (2016a). The weights are estimated using Maximal Likelihood Estimation (MLE). Besides, the final sentence vector is generated by subtracting the first principle component, which stands for the common discourse vector. This method shows the same speed as that of averaged sentence embeddings and better performance. It will be used as a baseline in this paper. Ethayarajh (2018) extended SIF to eliminate the requirement of hyper-parameters using an angular distance-based random work model. Inspired by the Gram-Schmidt Process, Yang et al. (2019) builds the sentence embedding from the orthogonal basis of word embeddings. By extending the arithmetic mean to the *power means* and concatenating *power means* of different word embeddings, Rücklé et al. (2018) narrows the gap between parameterized and non-parameterized methods.

B. Locally-Aggregated Sentence Embeddings

By taking the idea of Vector of Locally-Aggregated Descriptors applied in image representation Jégou et al. (2010), Vector of Locally-Aggregated Word Embeddings (VLAWE) Ionescu and Butnaru (2019) builds document/paragraph embeddings form the residual between word embeddings and centroids of clusters. Compared to VLAWE, instead of using the centroids, the probability from Gaussian Mixture Models(GMM) clustering is used as weights for each word in Sparse Composite Document Vectors (SCDV) Mekala et al. (2016). The sentence embedding is built upon the concatenation of weighted word embeddings summing over all words in a sentence. But both methods suffer from high dimensional problems ($k \times d$ where k is the number of clusters and d is the dimensionality of the word embedding).

C. Distance-based Sentence Embeddings

Nikolentzos et al. (2017) regards a document/paragraph as a multivariable Gaussian Distribution. The word embeddings contained by a document are used to estimate the mean and the covariance matrix of the distribution of a document. Therefore, the similarity of two documents can be measured as the distance between two distributions using KL Divergence etc. Compared with it, Torki (2018) and Wang et al. (2020) directly flatten the covariance matrix to be the representation of a document/sentence. Wu et al. (2018) proposed an unsupervised sentence embeddings by incorporating the idea of Word Mover's Distance (WMD) Kusner et al. (2015), which is a function that calculates the *travel cost/similarity* of two documents on the basis of the earth mover's distance metric (EMD) Rubner et al. (1998).

2.2 Parameterized Models

Deep contextualized language models like ELMo Peters et al. (2018), BERT Devlin et al. (2018) and GPT-3 Brown et al. (2020) set new state-of-the-art performances on various downstream tasks. They take word embeddings in a sentence as the input and output the contextualized word embeddings by using the attention mechanism or the bi-directional LSTM (BiLSTM) architecture. These models usually contain countless parameters (Fig 2.1), require many hours to train. GPT-3 has 1.75×10^{11} parameters. The simplest way to build sentence embeddings is to take the average of the output from these models. But, as reported in Reimers and Gurevych (2019) and Wang and Kuo (2020), the raw representation from BERT is not suitable for building sentence embeddings. Therefore, there are several modifications to the pre-trained BERT model.

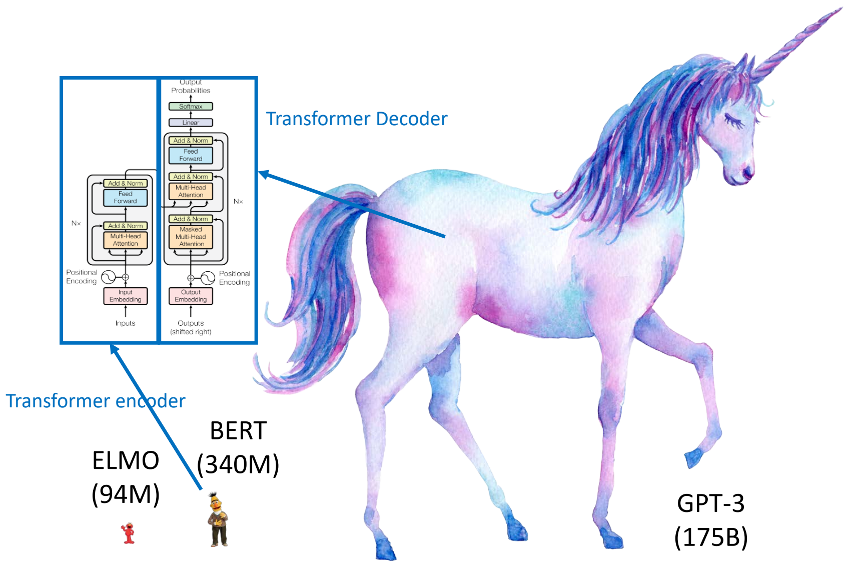


Figure 2.1: ELMo,BERT,GPT-3[1]

Reimers and Gurevych (2019) fine-tunes BERT by adding an additional classification objective function in a Siamese architecture. Wang and Kuo (2020) introduces a unified word representation for each word by integrating representations across layers of BERT. Then the sentence embedding is built using the weighted sum of unified word embeddings. It achieves state-of-the-art performance on semantic textual similarity tasks. Kiros et al. (2015) is an extension of the Skip-thought model introduced in Mikolov et al. (2013b). It trains an encoder-decoder framework using unlabeled triplets of sentences. After training, the model is frozen and the encoder is used to generate sentence embeddings. Conneau et al. (2017) trained a BiLSTM network using the supervised data of the Stanford Natural Language Inference(SNLI) datasets. Cer et al. (2018) trained the encoder of the Transformer Vaswani et al. (2017) architecture on SNLI. Generally, parameterized methods have better performance than non-parameterized methods. The power of deep contextualized language models is that they can be easily used by transfer learning and show surprising results on many downstream tasks.

This paper takes ideas of the locally-aggregated and distance-based sentence embeddings. We will show how our methods are linked with the essential ideas of deep contextualized language models in Chapter 4.

[1]From Hung-yi Lee: http://speech.ee.ntu.edu.twl

Chapter 3

Word Embeddings

Word embeddings are basic blocks for building sentence embeddings. In this chapter, we review mainstream word embedding models that will be used in chapters 4 and 5.

3.1 Word2Vec

The introduction of the Word2Vec model is a major step forward in NLP. Many word embedding models are built based on or fine-tune the Word2Vec. In this section, we explain two models proposed by Mikolov et al. (2013a) in detail: Continuous Bag-of-Words Model (CBOW) and Continuous Skip-Gram Model(SG).

Architecture. CBOW and SG are both shallow neural networks with one hidden layer. The input is a one-hot vector in which each component represents a real word in the vocabulary. The output of the network is still a vector that has the same size as the input. Each component of the output stands for a clear meaning: the probability of generating the current word given the input.

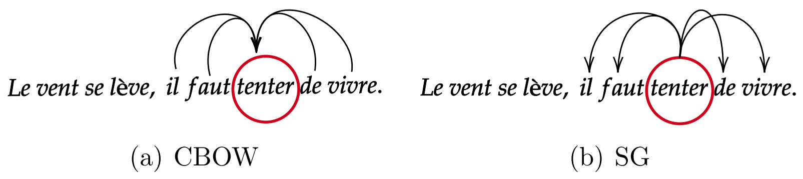

Le vent se lève, il faut tenter de vivre.
(a) CBOW
Le vent se lève, il faut tenter de vivre.
(b) SG

Figure 3.1: CBOW (a): given context words, predict the center. SG (b): given the center, predict context words.

Goal. CBOW and SG share the same objective function, which is to predict target words given the input words. The difference is that CBOW predicts the center word given its context, whereas SG is to predict context words given the center word as shown in Fig 3.1. There are two weight matrices: W input layer $\rightarrow$ hidden layer whose size is $|V| \times N$, where $|V|$ is the size of the vocabulary and N is the dimensionality of the hidden layer, M hidden layer $\rightarrow$ output layer whose size is $N \times |V|$. After training, the weight matrix W between the input layer and the hidden layer will be used to be the word embeddings.

The meaning of context words and center words are illustrated in Fig 3.2. When the window size is defined (say 2) and the center word is chosen, the context words are words covered by the window. If the center word is at the beginning, target words are those on the right.

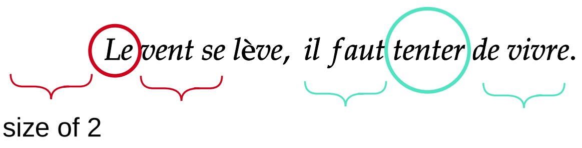


Figure 3.2: Given the center word *Le*, its contexts are *vent se*; Given the center word *tenter*, its contexts are *il faut* and *de vivre*.

CBOW: Continuous Bag-of-Words

In the following, we will use V to denote the size of the vocabulary and N to be the dimensionality of the hidden layer. The first weight matrix is represented as W and the second matrix is represented as M. The input will be represented as X.

Fig 3.3 shows the structure of CBOW. The input is a one-hot vector, where each dimension denotes a real word in the vocabulary. The input has several 1s and others remain 0. Each 1 means the corresponding word being the context of the center word that will be predicted. The output is a $V \times 1$ vector, where each dimension in the output denotes the probability of the current word being the center word given the input.

Forward Propagation

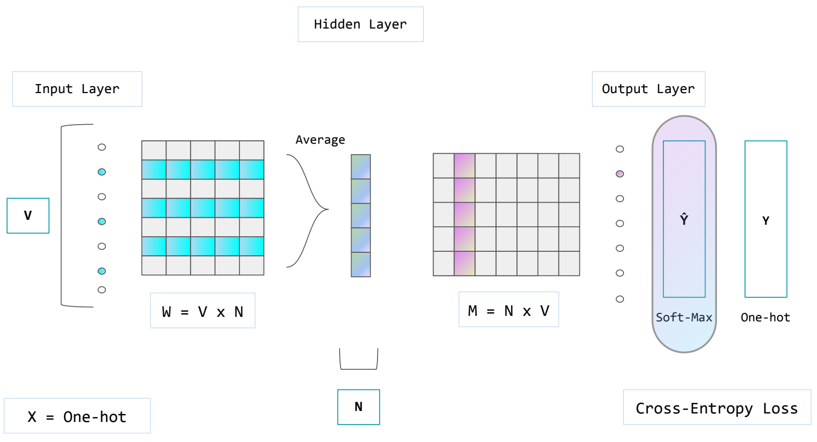


Figure 3.3: CBOW

From the input layer to the hidden layer. The input X is a special case of a one-hot vector, the propagation from the input to the hidden layer can be viewed as to take the corresponding rows from W and take the average of rows to get hidden units $(N \times 1)$.

$$h = \frac{1}{C}(X^T W) \tag{3.1}$$

From the hidden layer to the output layer. The hidden layer does the inner product with each column in M $(V \times N)$ to get the output vector u ($V \times 1$). Then, the output vector goes through the Softmax layer to become a probability distribution $\hat{y}$. $\hat{y}_i$ means the probability of word w_i being the center word given the context.

$$\hat{y}_{center} = p(w_{center}|w_{c,1}, w_{c,2}, \ldots, w_{c,C}) = \frac{exp(h^T M_{center})}{\sum_j^V exp(h^T M_j)} = \frac{exp(u_{center})}{\sum_j^V exp(u_j)} \tag{3.2}$$

In 3.2, we use M_{center} to denote the corresponding column that stands for the real center word in M, use u_{center} to denote the inner product of h and M_{center}.

What are the objective function and labels?. The label is also a one-hot vector (Y in Fig 3.3). After collecting the data (sentences), the center word and the context can be decided. In CBOW, the one-hot vector of the center word is the label. Thus the objective function can be reasonably defined as to maximize the probability of getting the real center word given the context. It is equal to minimize the negative of the objective function. In Eq 3.3, $\mathcal{L}$ is used to denote the loss function.

$$\begin{aligned} Obj &: \max p(w_{center}|w_{c,1}, w_{c,2}, \ldots, w_{c,C}) \\ &= \max \hat{y}_{center} = \max log \hat{y}_{center} = \max log \frac{exp(u_{center})}{\sum_j^V exp(u_j)} \\ &\Rightarrow \mathcal{L} = \min log \sum_j^V exp(u_j) - u_{center} \end{aligned} \tag{3.3}$$

Why the loss function $\mathcal{L}$ can be seen as a special case of the cross-entropy loss? The label of a pair of context words and a center word is a one-hot vector, which is also a distribution. The output is also a distribution after going through the Softmax layer. The objective is to minimize $-log\hat{y}_{center}$, as shown in 3.4, it can be expanded as a cross-entropy loss.

$$\begin{aligned} \min -\log \hat{y}_{center} &= -0 \times \log \hat{y}_0 - 0 \times \log \hat{y}_{center} \ldots - 1 \times \log \hat{y}_{center} \ldots - 0 \times \log \hat{y}_V \\ &= -y_{center} \times \log \hat{y}_{center} \end{aligned} \tag{3.4}$$

Backward Propagation

Update M. Take the derivative of $\mathcal{L}$ with respect to M. In 3.5, M_j means the j_{th} column in M. $\mathbb{1}(j = center)$ is the indicator function that means this term is 1 $\Longleftrightarrow$ the current word is the real center word. Fig 3.4 illustrates this process.

$$\frac{\partial \mathcal{L}}{\partial M_j} = \frac{\partial \mathcal{L}}{\partial u_j} \cdot \frac{\partial u_j}{\partial M_j} = \underbrace{\hat{y}_j - \mathbb{1}(j = center)}_{scalar:error\ term} \cdot h \tag{3.5}$$

Update W. In 3.6, w_j means the i_{th} row of the first weight matrix W. Fig 3.5 illustrates the process.

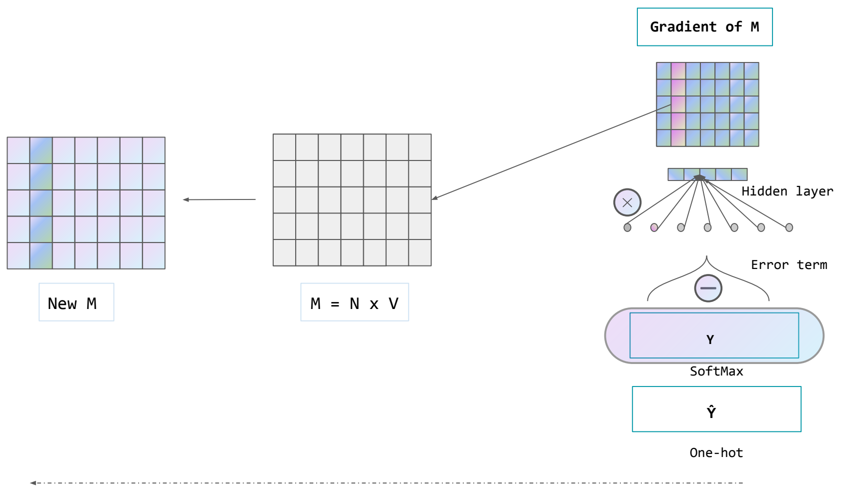


Figure 3.4: Take the derivative w.r.t. j_{th} output (w_j is the word in the vocabulary). It leads to a scalar that is used to do the product with h to update one row in M whose column index is the same as the index of w_j.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W} &= \frac{\partial \mathcal{L}}{\partial h} \cdot \frac{\partial h}{\partial W} \\
&= [\sum_{j}^{V} \frac{\partial \mathcal{L}}{\partial u_j} \frac{\partial u_j}{\partial h}] \cdot \frac{\partial h}{\partial W} \\
&= [\sum_{j}^{V} \underbrace{\hat{y}_j - \mathbb{1}(j = center)}_{scalar:error\ term} \cdot \underbrace{M_j}_{size:1\ \times\ N}] \cdot \frac{\partial h}{\partial W} \\
&= x \times [\sum_{j}^{V} \hat{y}_j - \mathbb{1}(j = center) \cdot M_j]^T
\end{aligned} \tag{3.6}$$

Skip-Gram

SG is to predict the contexts given a center word. It is essentially the same as CBOW. The output of the network is a vector ($V \times 1$). The objective function is to maximize the probability of predicting the correct context given the center word.

Forward Propagation

Compared with CBOW, instead of inputting multiple context words, the input in SG is a real one-hot vector of which one component is 1 and others are 0. Blue row in Fig 3.6 stands for the center word. The output is still a vector, but it models posterior probabilities of getting multiple context words given a center word as shown in Eq 3.7. Moreover, by taking the i.i.d. assumption, the posterior probability can be factorized.

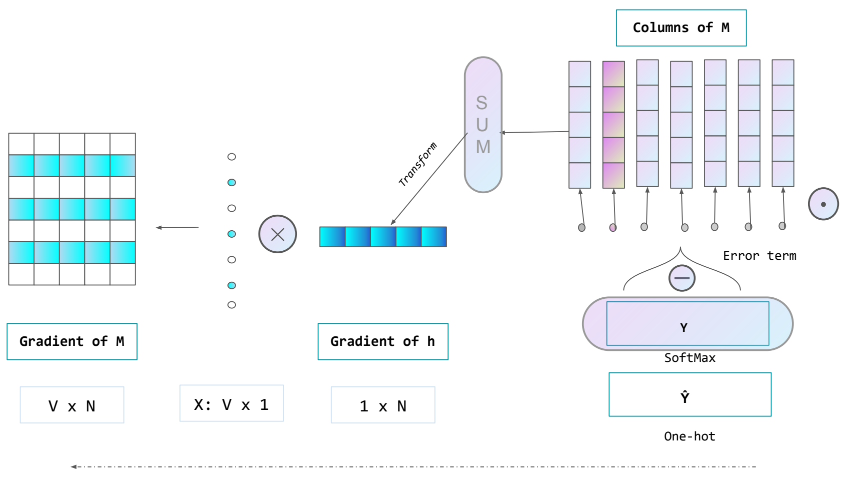


Figure 3.5: Take the derivative w.r.t. W. By applying the chain rule, $\frac{\partial \mathcal{L}}{\partial h}$ results in a $N \times 1$ dimensional vector. Then take this vector do the matrix product with x ending up with a $N \times V$ matrix, which is the gradient of W. Since X is a special one-hot vector, the update of W can be seen as to update multiple rows of W using the same gradient of h.

$$\begin{aligned} &p(w_{c,1}.w_{c,2},\dots,w_{c,C}|W_{center}) \\ &= \prod_{i}^{C} p(w_{c,i}|W_{center}) \end{aligned} \tag{3.7}$$

we could rewrite the objective function by following the same strategy in CBOW to maximize the probability. In Eq 3.8, we use C to denote the number of context words.

$$\begin{aligned} Obj: \max &\prod_{i}^{C} p(w_{c,i}|W_{center}) \\ &= \max \sum_{i}^{C} \log p(w_{c,i}|W_{center}) = \max \sum_{i}^{C} \log \frac{exp(u_i)}{\sum_j^V exp(u_i)} \\ &\Rightarrow \mathcal{L} = \min -\sum_{i}^{C} u_i + C \times log \sum_{j}^{V} exp(u_j) \end{aligned} \tag{3.8}$$

In order to simplify Fig 3.6, only two context words are assumed which means there are two panels in the output. Each has a $V \times 1$ output vector. Different panels have its own labels.

Backward Propagation

Update M. Compared with CBOW, instead of outputting one vector, SG outputs C vectors, each of which has its own labels. In Eq 3.9, M_j stands for the j_{th} column of M.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial M_j} &= \frac{\partial \mathcal{L}}{\partial u_j} \cdot \frac{\partial u_j}{\partial M_j} \\ &= C \times u_j \times \mathbb{1}(jincontext) \end{aligned} \tag{3.9}$$

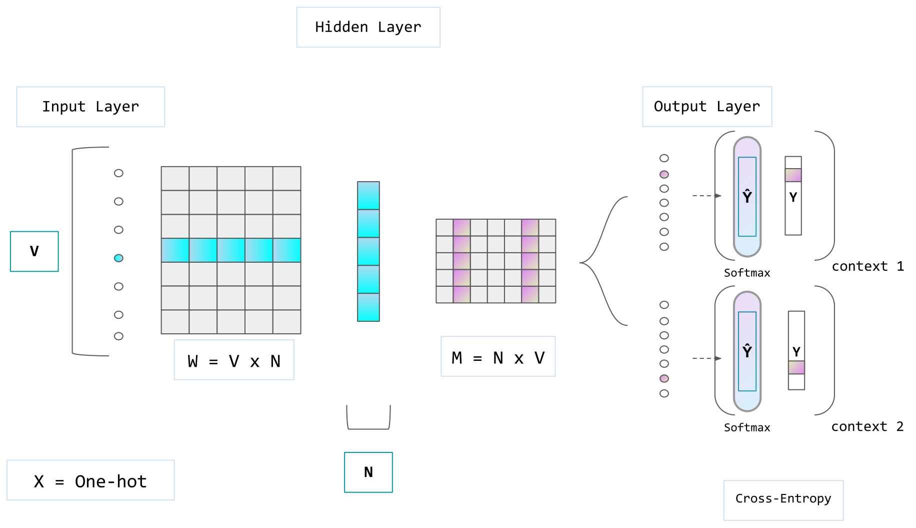


Figure 3.6: The row in blue of W stands for the product of the input one-hot vector and W. The three columns in color means two real context words.

In Fig 3.9, as there are C context words, the only difference is the error term. Compared with CBOW, instead of computing one error vector, SG needs to calculate C error vectors $(C \times V)$ then sum over C to get an $N \times 1$ vector (error term).

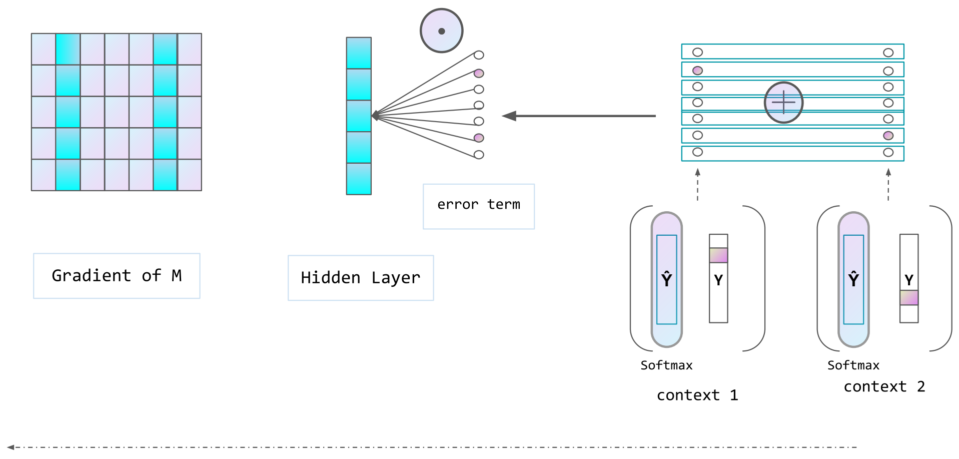


Figure 3.7: Take the derivative w.r.t. each component in the output vector in each panel.

Update W. After constructing the error term vector in Fig 3.7, the update of W is the same as that in CBOW. SG only updates one row in W each time that stands for the center word.

3.2 GloVe

Compared with Word2Vec, Glove Pennington et al. (2014) utilizes the global statistics of the corpus from the co-occurrence matrix of data.

Co-occurrence Matrix

The co-occurrence matrix is a V^2 document matrix, where V is the size of the vocabulary. The rows and the columns represent words. The element in the i_{th} row and the j_{th} column denote the number of times that $word_j$ being the context of $word_i$. For example, if we have a sentence *Le vent se lève*, the co-occurrence matrix with the window size of 1 can be constructed as Tab 3.2.

Count Words Words	Le	vent	se	lève
Le	0	1	0	0
vent	1	0	1	0
se	0	1	0	1
lève	0	0	1	0

Table 3.1: The co-occurrence matrix of the sentence *Le vent se lève*

Glove Embeddings

Based on the co-occurrence matrix, co-occurrence probabilities can be constructed. Let X_{ij} be the number of times that $word_j$ appears in the context of $word_i$. Then let $X_i = \sum_j X_{ij}$ be the total count of words being the context of $word_i$. Let $P_{ij} = \frac{X_{ij}}{X_i}$ be the probability of the $word_j$ being the context of $word_i$. GloVe vectors are built based on the assumption: given an intermediate word w_k, if w_k is related to w_i but not related to w_j, then the ratio of $\frac{P_{ik}}{P_{jk}}$ should be large. If w_k is more related to w_j than w_i, then the ratio of $\frac{P_{ik}}{jk}$ should be small.

$$\mathcal{F}(w_i, w_j, w_k) = \frac{P_{ik}}{P_{jk}} \tag{3.10}$$

In Eq 3.10, w_i, w_j, w_k are word embeddings to learn. Eq 3.10 models the relationship of a triplet of words without the exact expression.The goal is to capture semantic differences of w_i and w_j whereas w_k serves as an intermediate word. So it is natural to consider using the difference of two words. Thus $\mathcal{F}$ can be rewritten as Eq 3.11.

$$\mathcal{F}(w_i - w_j, w_k) = \frac{P_{ik}}{P_{jk}} \tag{3.11}$$

Since the left side of Eq 3.11 is a function of two vectors and the right side is a scalar, the expression of $\mathcal{F}$ can be defined as the inner product of two vectors:

$$\begin{aligned} \mathcal{F}((w_i - w_j)^T w_k) &= \frac{P_{ik}}{P_{jk}} \\ \mathcal{F}(w_i^T w_k - w_j^T w_k) &= \frac{P_{ik}}{P_{jk}} \end{aligned} \tag{3.12}$$

How to link the left and the right in Eq 3.12? The left is a function of the difference of two vectors and the right is a fraction. We could define $\mathcal{F}$ as an exponential function to link both sides.

$$\exp(w_i^T w_k - w_j^T w_k) = \frac{\exp(w_i^T w_k)}{\exp(w_j^T w_k)} = \frac{P_{ik}}{P_{jk}} \tag{3.13}$$

Eq 3.13 reveals the relationship of $exp(w_i^T w_k)$ and P_{ik}, which can be used to derive the loss function of GloVe. Take the $\log$ on both sides, we could get Eq 3.14

$$\begin{aligned} \exp(w_i^T w_k) &= P_{ik} = \frac{X_{ik}}{X_i} \\ \Rightarrow w_i^T w_k &= \log(x_{ik}) - \log(x_i) \end{aligned} \tag{3.14}$$

As the left side is not sensitive to the swap of w_i and w_k, Pennington et al. (2014) introduces two bias terms b_i and b_k to alleviate this problem. Since $\log(x_i)$ is a scalar, it can be included in b_i. Eq 3.14 can be written as: $x_i^T x_k = \log(x_{ik}) + b_i + b_k$. The loss function can be defined as to minimize the difference between $w_i^T w_k$ and $\log(x_{ik}) + b_i + b_k$ in the whole corpus.

$$\mathcal{L} = \sum_{i,j}^{V} f(x_{ij})(w_i^T w_k + b_i + b_k - \log(x_{ik})) \tag{3.15}$$

Training. w_i, w_k, b_i, b_k are parameters to learn. $\log(x_{ik})$ can be considered as labels collected from data. In Eq 3.15, V is the size of the vocabulary and $f(X_{ij})$ is the weight for $word_i$ and $word_j$.

3.3 FastText

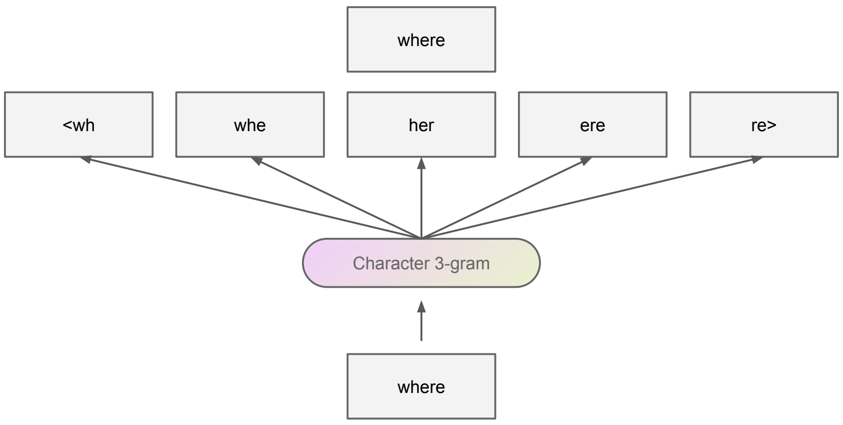


Figure 3.8: character 3-gram words of *where*.

FastText Bojanowski et al. (2017) is the extended model of Skip-Gram (SG) in Word2Vec using negative sampling. But FastText considers the morphological structure of words. It splits a word into character n-grams as illustrated in Fig 3.8, where n ranges from 3 to 6. The word itself is also included in its n-gram subwords. In Fig 3.8, $<$ and $>$ are special symbols representing the boundary of a word. Each subword learns its own representation. By incorporating subwords, FastText is able to handle OOV (out-of-vocabulary) words, which is simply to average representations of its n-grams.

The loss function of FastText is the negative log-likelihood function with negative sampling introduced in Mikolov et al. (2013b). ℓ in Eq 3.16 is the logistic loss: $\ell(x) \rightarrow 1/(1+\exp(-x))$. $s\left(w_t, w_c\right)$ means a scoring function between a word w_t and a word w_c. It is natural to define a scoring function to be the inner product of two vectors.

$$\sum_{t=1}^{T}\left[\sum_{c \in \mathcal{C}_t} \ell\left(s\left(w_t, w_c\right)\right)+\sum_{n \in \mathcal{N}_{t,c}} \ell\left(-s\left(w_t, n\right)\right)\right] \tag{3.16}$$

3.4 LexVec

PMI matrix and Positive PMI matrix

Point-wise Mutual Information (PMI) matrix is constructed from a co-occurrence matrix. The entry of PMI matrix PMI(x,y) measures the match of a word w and a context c. In Eq 3.17, p(x,y) is the joint probability of a word x and a word y occurring together. The entry of PMI ranges from $-\infty$ to $+\infty$ due to $P(x,y)=0$. To address this problem, Positive PMI (PPMI) is proposed. It replaces negative values with 0.

$$\begin{aligned} PMI(x,y) &= \log \frac{P(x,y)}{P(x)P(y)} \\ PPMI(x,y) &= \max\left(\log_2 \frac{P(x,y)}{P(x)P(y)}, 0\right) \end{aligned} \tag{3.17}$$

LexVec Vectors

Levy and Goldberg (2014) shows that the Skip-Gram model with negative sampling (SGNS) is implicitly doing the factorization of a Shifted PMI matrix. LexVec is also built upon the idea of the matrix factorization but with a PPMI matrix. It tends to minimize the loss in Eq 3.18 where W_w and W_c are vector representations that will be learned Salle et al. (2016).

$$\begin{aligned} L_{wc}^{LexVec} &= \frac{1}{2}\left(W_w \tilde{W}_c^{\top} - PPMI_{wc}^{*}\right)^2 \\ L_{w}^{LexVec} &= \frac{1}{2}\sum_{i=1}^{k} \mathbf{E}_{w_i \sim P_n(w)}\left(W_w W_{w_i}^{\top} - PPMI_{ww_i}^{*}\right)^2 \end{aligned} \tag{3.18}$$

3.5 PSL

Paraphrase Database (PPDB) Ganitkevitch et al. (2013) is a collection of phrase pairs rated by its similarity. PSL Wieting et al. (2015) is trained on word pairs extracted from PPDB using a recursive neural network (RNN) architecture. The loss function is the same as that in Skip-Gram Mikolov et al. (2013b) with negative sampling. PSL embeddings are initialized by Skip-Gram embeddings. Because of the nature of PPDB, PSL is good at capturing the semantic differences of words.

3.6 Contextualized Word Embeddings

The aforementioned word embeddings are static. One word has an exact representation. As for contextualized word embeddings, it builds representations for each token, which means one word could have several representations depending on its context. For example, consider a polysemous word *bank*. When we have a phrase *river bank* and a phrase *commercial bank*, the word *bank* has different meanings. It is reasonable to assign them different representations. ELMo Peters et al. (2018) and BERT Devlin et al. (2018) are two representative contextualized language models.

ELMo

Peters et al. (2018) is an unsupervised language model using the bi-directional LSTM architecture (biLM). The input is a sentence. The output is a sequence of embeddings corresponding to each token in the sentence. It is trained in an unsupervised manner. Given a sentence, a forward model and a backward model are conducted simultaneously to predict the next word given the current word. The representation of each token is built by concatenating the hidden layers. But in biLM, there are multiple hidden layers, thus one token could have multiple embeddings. ELMo takes the weighted sum of these embeddings to be the final representation of the token. The weights are learned by downstream tasks, which means the token could have different representations for different tasks. Fig 3.9 shows the architecture of ELMo.

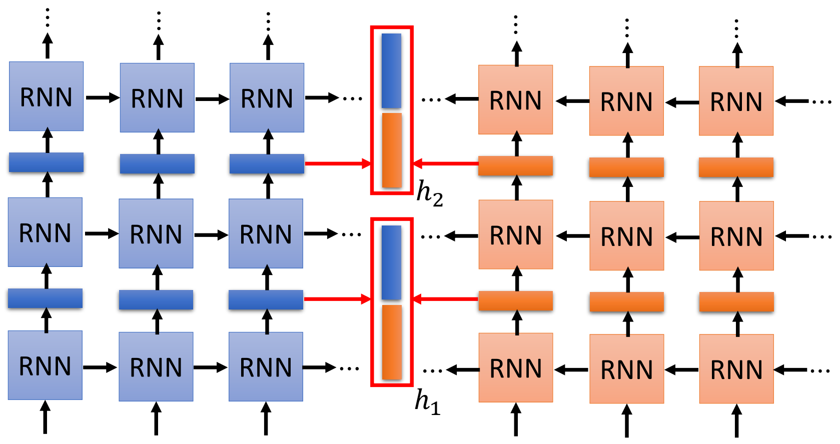


Figure 3.9: ELMo biLM architecture [1]

Transformer

Transformer is a sequence-to-sequence model proposed by Vaswani et al. (2017) with the encoder and the decoder. Its architecture is illustrated in Fig 3.10. By taking the advantages of attention mechanism, the encoding can be computed in parallel. Therefore, it is used to replace bi-directional LSTM model.

Encoder Self-Attention. The attention mechanism is essentially the weighted sum of the context given one center word. Given one embedding of the center word, first, it is transformed

[1]From Hung-yi Lee: http://speech.ee.ntu.edu.tw/t̃lkagk

into 3 vectors called: *Query vectors*(Q), *Key vectors*(K) and *Value vectors*(V). The Q of the center word is used to do the inner product with all other words' K including its own. It leads to a list of scalars, each of which can be considered as the *Importance* of two words. Then the *Importance* are treated as weights and are used to multiply the corresponding V to get the weighted sum representation of the center word.

Decoder Self-Attention. The decoder is to generate the next word given the current word. Therefore, the words behind the current word are masked. The current word only does the attention with its preceding words.

Encoder-Decoder Attention. In this layer, the current word only does the attention with the output of the encoder.

BERT

BERT Devlin et al. (2018) is the encoder of the Transformer. The input is sentences and the output is a sequence of contextualized representations corresponding to each token in sentences. It is trained on two tasks. One is the Masked Language Model(MLM). 15 % words of a sentence are masked. The goal is to use the output of masked words to predict its original word. The other is called the sentence classification task. The input is two sentences separated by a special token *[SEP]*. The embedding of the beginning token *[CLS]* is used to predict if two sentences are a real pair or not.

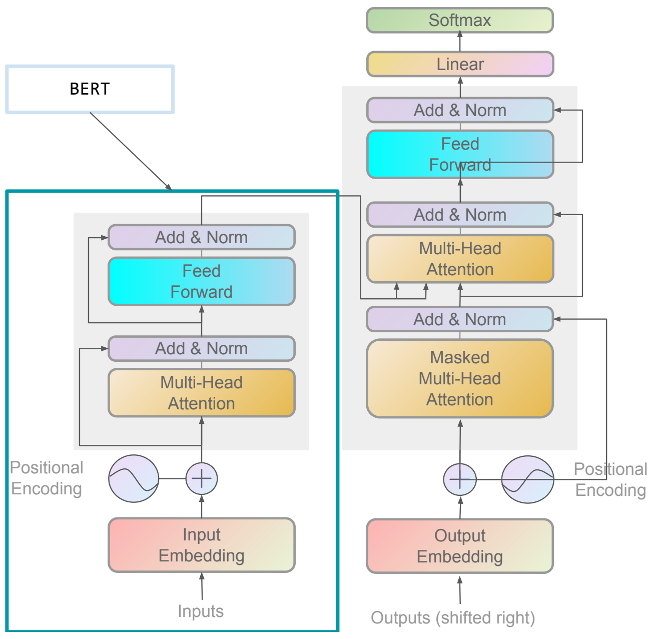


Figure 3.10: Transformer and BERT

Chapter 4

Methods

In this chapter, we present our method to generate sentence embeddings. Let $\mathbf{S} = \{s_1, \ldots, s_n\}$ be collected sentences. Our goal is to map $\forall s \in \mathbf{S}$ to be a vector representation. We hope that the cosine similarity of two sentences who share a similar semantic meaning is high whereas the cosine similarity of two sentences with different semantic meanings is low.

Fig 4.1 illustrates the general process. In 4.1, we have 4 sentences. So $|\mathbf{S}| = 4$ and $s_1 =$ *How old are you*, $s_2 =$*What is your age*, $s_3 =$ *My phone is good*, $s_4 =$ *Your cellphone looks great*. We expect representations of sentences could capture semantic relationships among them, which means similarities of s_1, s_2 and s_3, s_4 are high but similarities of s_1, s_3 and s_2, s_4 are low. The aforementioned is what shown in the heatmap of 4.1.

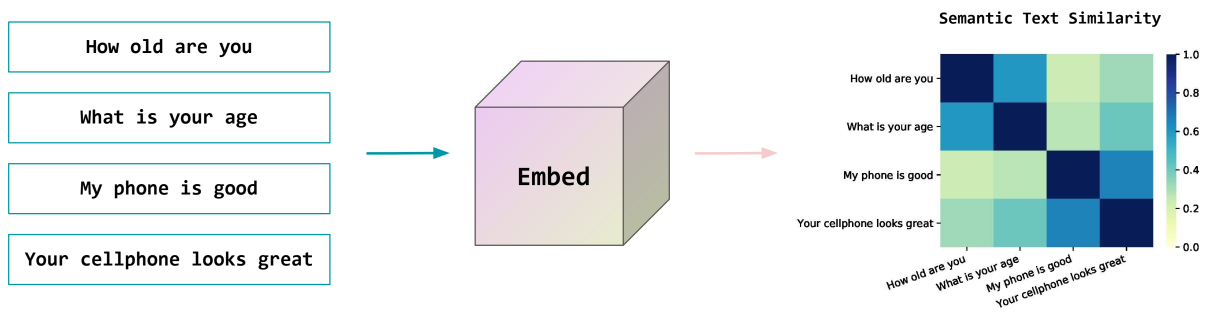


Figure 4.1: Cosine similarities calculated using our embeddings.

Pre-processing. In this step, collected sentences are split into words. Each word is lowercased and will be assigned a unique index. Special characters (presumably @,*,etc.) are removed. A lookup matrix that transforms a one-hot vector to its word embedding will be created. Depending on different word embeddings, the lookup matrix may vary. This part will be discussed in the experimental setting. Finally, We end up with the vocabulary extracted from S, two dictionaries mapping a word to its index and mapping an index to a word, a lookup matrix.

Estimate weights. In this step, we introduce a weight for each word. It is the same setting in Arora et al. (2016b). Weights are introduced to reduce the impact of words that occur frequently regardless of the context (presumably "the", "a " etc). The weight is estimated by the inverse of its unigram probability p in the corpus as shown in Eq 4.1. To avoid zero, p is estimated using Laplace Smoothing. We use $\#(word_m)$ to denote the count of the $word_m$, use N to denote the total number of words, and V is the size of the vocabulary.

$$p_m = \frac{1 + \#(word_m)}{N + V}, weight_m = \frac{1}{p_m} \tag{4.1}$$

4.1 Universal Sentence

In this step, we will use locally-aggregated word embeddings to model *Universal Sentence*.

First, we convert each word in the vocabulary into its word embeddings resulting in a $V \times k$ matrix, where V is the size of the vocabulary and k is the dimensionality of word embeddings. k depends on the type of word embeddings you load. In GloVe, $k = 300$. In BERT, $k = 768$. We can consider each word as a data point with k dimensions. Next step is to cluster V words. Clustered word embeddings are used to model *Universal Sentence*. There are multiple options to choose a clustering algorithm. For word embeddings, it is common to use Kmeans to be the clustering algorithm (Wang et al. (2020), Ionescu and Butnaru (2019), Torki (2018), Butnaru and Ionescu (2017)). In this section, we argue that Kmeans is not proper for clustering word embeddings by means of the Zipf's law.

Zipf's Law. Zipf's law Zipf (1949) is an experimental law stating that the frequency that a word appears in a natural language corpus is inversely proportional to its rank. That is to say, the word with the highest frequency occur three times as often as the word with the third-highest frequency:

$$F_n = \frac{1}{n} * F_1 \tag{4.2}$$

where n is the rank of the $word_n$. F_n is the frequency of the $word_n$. F_1 is the highest frequency of the $word_1$ in the corpus.

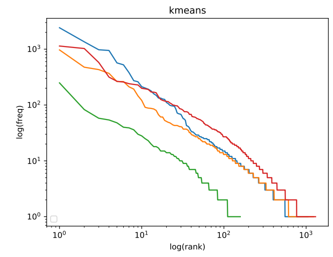
Problems with Kmeans

As reported in Martinet (2014), clustered word embeddings within each cluster still follow the Zipf's law. We posit that well-constructed clusters have their own functional roles. To explain it well, let us assume a simplified situation. All sentences consist of three blocks: (1) who (2) did what (3) to whom. In this case, we could set the number of clusters to be $3 + 1$, where the additional 1 cluster comes from the idea of *common discourse* words introduced in Arora et al. (2016b).

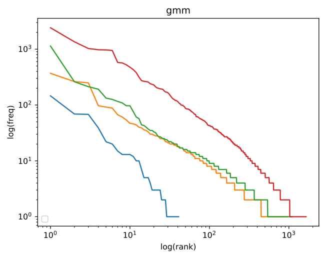
Ideally, after clustering, the first centroid is to model *who*, the second cluster is to model *did what*, and the last cluster is responsible for modeling *to whom*. Therefore, centroids can be considered as abstract blocks modeling general meanings above all sentences (hence the name Universal Sentence). Relationships between the current sentence and the cluster centroids can be used to distinguish the semantic differences of sentences.

We argue that Kmeans is not a suitable algorithm to cluster word embeddings for two reasons.

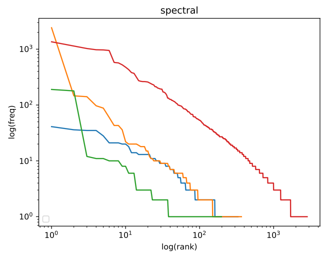
Curse of dimensionality. The target of the Kmeans algorithm is to minimize the intra-group distance. But when it comes to a high dimensional space, the distance measuring tends to be inflated. For word embeddings, they are usually to be high dimensional spaces ranging from 300 to 900.



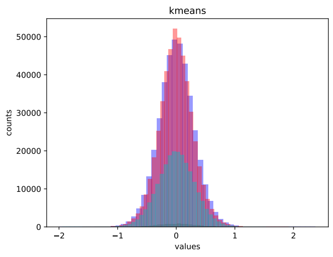
(a) Zipf's Law: Kmeans



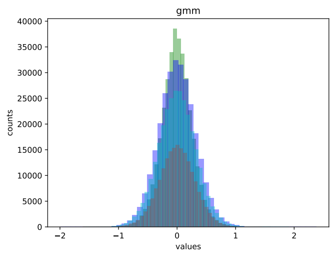
(b) Zipf's Law: GMM



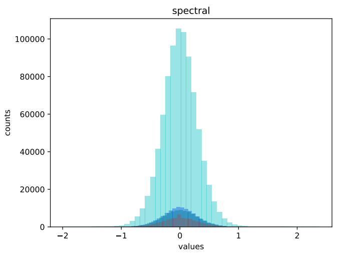
(c) Zipf's Law: Spectral



(d) value-frequency: Kmeans



(e) value-frequency: GMM



(f) value-frequency: Spectral

Figure 4.2: Zipf's Law distributions and value-frequency distributions in clusters. The X axis in the Zipf's Law distribution is the logarithmic rank of a word and the Y axis is the logarithmic frequency of a word. Different colors stand for different clusters. For the value-frequency distributions, we flatten word embeddings in the same cluster and plot the value distribution.

Ill-formed clusters. Martinet (2014) shows that Kmeans tends to equally assign words to clusters. This nature prevents centroids from modeling *Universal Sentence* by assigning some irrelevant words to the cluster. We illustrate this in Fig 4.2. To verify our assumption, We use the CR (customer review) dataset[1]. We filter the CR dataset that contains 2412 sentences (26824 words) to make it contain only simple sentences based on the length of sentences. The length is limited to be 15. After processing, the dataset contains 952 sentences, 9209 words. We test 3 different clustering algorithms: Kmeans, Gaussian Mixture Model(GMM), and Spectral Clustering Ng et al. (2002).

The number of clusters is set to be 4. We plot the corresponding Zipf's law distribution within each cluster and the value-frequency distribution for each algorithm. Arora et al. (2016b) shows that *common discourse* block is roughly the same for all sentences. It contains the most common words regardless of the syntax. We posit that the cluster that contains most frequent words stands for the *common discourse* cluster[2] . Thus, the Zipf's law distribution of the *common discourse* cluster should have the highest frequencies compared with other clusters.

In a of Fig 4.2, We could observe that three of Zipf's law distributions get entangled in Kmeans and the green one has the different Zipf's law distribution. It is hard to distinguish the *common discourse* cluster from these four Zipf's law distributions. It is not reasonable to consider one of three entangled Zipf's law distributions as the *common discourse* cluster

[1]https://nlp.stanford.edu/∼sidaw/home/projects:nbsvm

[2]Here the *common discourse* cluster means the cluster that models *common discourse*. So as to *did what* cluster, etc.

because words in those three clusters all share similar frequencies. But in our dataset, not all sentences have the format *who* + *did what* + *to whom*. For example, *I am happy* only contains components: $I \in$ *who*, *am* $\in$ *common discourse* and *happy* $\in$ *did what*. Hence, ideally, the *who* cluster, the *did what* cluster, and the *to whom* cluster should have lower Zipf's law distributions compared with that of the *common discourse* cluster. The Zipf's law distribution in green can not be treated as the *common discourse* cluster neither as it has lower frequencies.

As for GMM and Spectral Clustering, the *common discourse* cluster is more distinguishable (the red line in b and c of Fig 4.2) as it shows higher frequencies than others. In the values-frequency distribution of Spectral Clustering, this phenomenon is more obvious. One cluster contains many more words than others as shown in f of Fig 4.2.

Moreover. We manually extract several common words and show their distributions in Tab 4.1. We could see that Kmeans assign these common words into different clusters. It goes against our expectations. For example, since we want to model *common discourse* block, we do not want *a* and *an* belonging to different clusters. But in GMM and Spectral Clustering, the words are more concentrated into one cluster.

Words / Algorithms / Clusters	**Kmeans**	**GMM**	**Spectral**
Cluster 0	of, or	or	/
Cluster 1	a, this, with	/	a, the, an, and, in, this, with, or, on
Cluster 2	the, and, for, in, on, an	the, and, a, an, in, on, for, this, with, of	of, for
Cluster 3	/	/	/

Table 4.1: Common discourse words clustered by Kmeans, GMM and Spectral Clustering

4.2 Covariance matrix and Sentence Embeddings

In this step, we will construct a covariance matrix from clustered word embeddings and use it to represent a sentence.

Given two sentences, we expect vector representations to detect semantic differences of two sentences. For a given sentence S that includes words $\{w_1, w_2, \ldots, w_m\}$, each word will be assigned to a cluster after clustering. We first construct a weighted accumulated residual matrix $\Phi(S)$. For each centroid c_i of clusters, we have Eq 4.3. In the following, we will call v residual clusters.

$$v_i = \sum_{m \in S \cap C_i} weight_m \times (w_m - c_i) \tag{4.3}$$

In Eq 4.3, $weight_m$ is the estimated unigram probability of the word w_m, and c_i is the centroid of the i_{th} cluster. $m \in S \cap C_i$ means words in S that belong to the i_{th} cluster.

We end up with a $M \times k$ matrix in Eq 4.4, where M is the number of clusters and k is the dimensionality of word embeddings.

$$\Phi(S) = \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_M^T \end{bmatrix} = \begin{pmatrix} v_{11} & \dots & v_{1k} \\ v_{21} & \dots & v_{2k} \\ \vdots & \ddots & \vdots \\ v_{M1} & \dots & v_{Mk} \end{pmatrix}_{M \times k} \tag{4.4}$$

$\Phi(S)$ measures the differences between the given sentence and *Universal Sentence*. We will then use the joint variability of two random residual clusters in $\Phi(S)$ to build a sentence embedding, which is the covariance matrix.

Torki (2018) shows that the covariance matrix can capture the distinctness of word embeddings. They construct the covariance matrix based on each dimension of word embeddings hoping to model the dissimilarities of two documents/paragraphs.

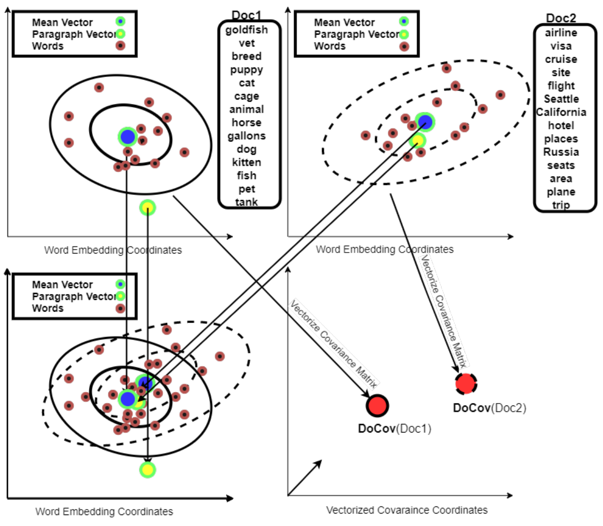


Figure 4.3: Torki (2018)

Fig 4.3 illustrates the covariance matrix of word embeddings using confidence ellipse representations. Top left and top right are two sets of word embeddings represented by its first two dimensions and their covariance matrix representations. The bottom left is the mix of two figures and the bottom right is document embeddings represented by the first two dimensions of the flattened covariance matrix.

We hope that sentences with similar semantic meanings could show similar variability for each pair of residual clusters. These unique relationships can be used to distinguish semantic differences in sentences.

The covariance matrix can be derived from $\Phi(S)$. Each row in $\Phi(S)$ is treated as a data point, each of which has M dimensions. The entry of the covariance matrix is shown in Eq 4.5.

$$\sigma_{ij} = \mathbb{E}[(\Phi(S)_i - \mathbb{E}_i)(\Phi(S)_j - \mathbb{E}_j)] \tag{4.5}$$

$\mathbb{E}_i$ is the mean of the i_{th} row. $\Phi(S)_i$ means the i_{th} row. We end up with a $M \times M$ matrix. The final sentence embedding $\mathcal{V}_S$ is built by flattening the upper triangular of C. $\mathcal{V}_S = \{\sigma_1^2, \sigma_{12}, \dots, \sigma_2^2, \sigma_{2M}, \dots, \sigma_M^2\}_{1 \times m(1+m)/2}$.

$$C = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \dots & \sigma_{1M} \\ \sigma_{12} & \sigma_2^2 & \dots & \sigma_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{M1} & \sigma_{M2} & \dots & \sigma_M^2 \end{pmatrix}_{M \times M} \tag{4.6}$$

Normalization. Two kinds of normalization are applied to $\mathcal{V}_S$. First, the power normalization x^α is applied to each component of $\mathcal{V}_S$. Jegou et al. (2011) shows that the optimal value α depends on the number of clusters. But, $alpha = 0.5$ is a near-optimal setting for $M = 16$ to $M = 256$. The power normalization is used for reducing the influence of highly frequent words. Then, the vector is L_2 normalized[3] to get the final representation. These two steps are the same as in Jegou et al. (2011). The pipeline of the algorithm is shown in Algo 1.

Algorithm 1 Compute the sentence embeddings given sentences S

```
Input: A set of sentences
  A pre-computed lookup table to convert words into word embeddings
  Build the vocabulary with the size of $V$ from sentences
  Estimate unigram probabilities for $V$ words using Maximal Likelihood Estimation
  Clustering the vocabulary into $M$ clusters and calculate centroids

  for sentence in sentences do
      Split the sentence into $n$ words. $S = \{w_1, \dots, w_n\}$
      Convert words into a $n \times k^4$ word embedding matrix
      Build the $M \times k$ accumulated residual matrix
      Build the $M \times M$ covariance matrix and flatten the up triangle as a $1 \times \frac{M(1+M)}{2}$ vector $\mathcal{V}_S$
      Normalized and get the sentence embedding
  end for
  return a set of sentence embeddings
```

4.3 Embedding Distribution and Self-attention

In this sections, we will review the Self-attention mechanism and further discuss the relationship between our method and self-attention.

The self-attention mechanism is introduced in the Transformer architecture Vaswani et al. (2017). It is to make the current word focus on the more important information provided by its context. It is essentially doing the inner product with its context. The result denotes how important the context is for the current word. The representation of the current word is built by the weighted sum of its context embeddings.

Fig 4.4 is the visualization of self-attention. When the network is dealing with words *more* and *difficult*, most weights are applied to the word *making* to complete the phrase *making . . . more difficult*. Different colors in 4.4 represent different heads. The representation of *more* and *difficult* is largely depends on the representation of *making*.

We built the sentence representation based on the accumulated residual between word embeddings and centroids. The covariance matrix measures the joint variability of each pair of residual clusters. The final embeddings are constructed by the upper triangular of the matrix since it is symmetric. If the value of σ_{C_1,C_2} is high, it means the variability of the residual *Cluster*$_1$ and the residual *Cluster*$_2$ is highly related. If σ_{C_1,C_2} is low, then the residual *Cluster*$_1$ and the residual *Cluster*$_2$ are less related. In conclusion, a sentence embedding is featured by inner intersections of residual clusters.

[3] $\mathcal{V}_S = \frac{\mathcal{V}_S}{||\mathcal{V}_S||_2}$

[4] k is the dimensionality of loaded word embeddings.

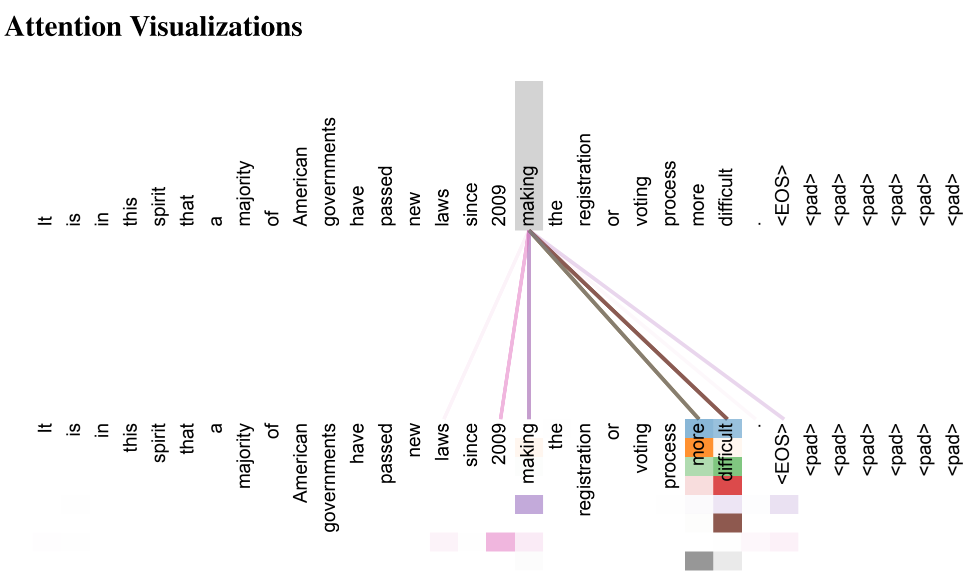


Figure 4.4: Attention Visualization Vaswani et al. (2017)

Our expectations for sentence embeddings are similar to the self-attention mechanism. Self-attention allows current word embeddings to largely depend on its important context. We expect that semantic similar sentence embeddings could both have similar skewed value distributions. In other words, we try to maximize the differences among sentences with different semantic meanings. For example, for sentences *What is your age* and *How old are you*, we expect to see that σ_{ij} to be relatively high or relatively low compared with other dimensions. Therefore, the differences of semantic different sentences are more distinguishable.

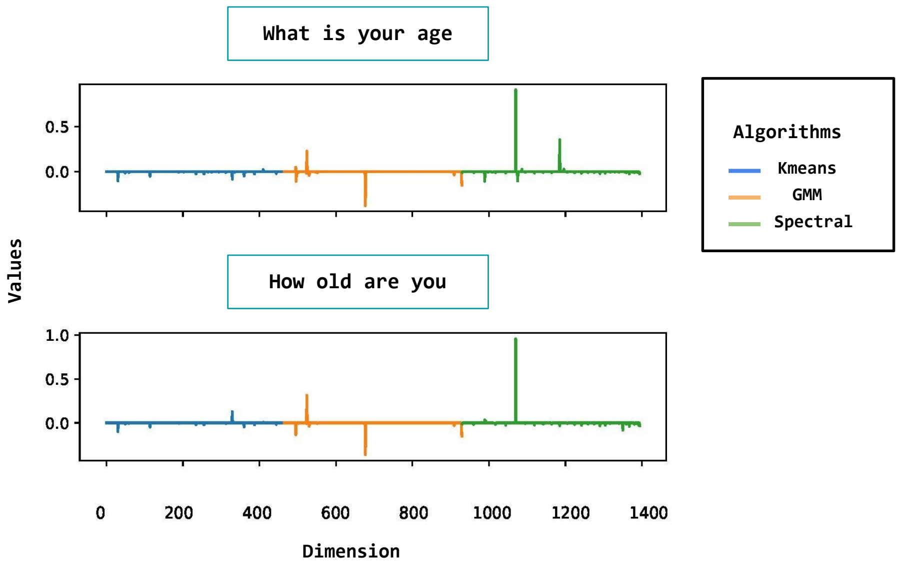


Figure 4.5: Embedding Distribution

Fig 4.5 is value distributions of sentences *What is your age* and *How old are you* built by Kmeans, GMM, and Spectral Clustering. We concatenate sentence embeddings of one sentence built by three algorithms. Different algorithms are marked by different colors. We could see that the values in *Kmeans* representation is almost uniform (blue segments in Fig 4.5) compared with that in *GMM* and *Spectral Clustering*. In *GMM* and *Spectral Clustering*, two sentence embeddings show similar patterns. More importantly, the vector is featured by sporadic dimensions.

Chapter 5

Results

In this chapter, we evaluate our method on Semantic Textual Similarity (STS) tasks to see if our method could capture semantic differences of sentences as expected.

Datasets. STS includes 6 datasets Agirre et al. (2012), Agirre et al. (2013),Agirre et al. (2014), Agirre et al. (2015), Agirre et al. (2016). Each STS dataset includes pairs of sentences collected from news, headlines and image descriptions, etc. STS compares semantic similarities of two sentences. It calculates the cosine similarity of sentence embeddings. The results are compared with human-labeled similarity scores ranging from 0 to 5 where 5 means two sentences are completely equivalent and 0 means two sentences are completely dissimilar. In the end, the average of Pearson correlations of computed similarities and labels are reported.

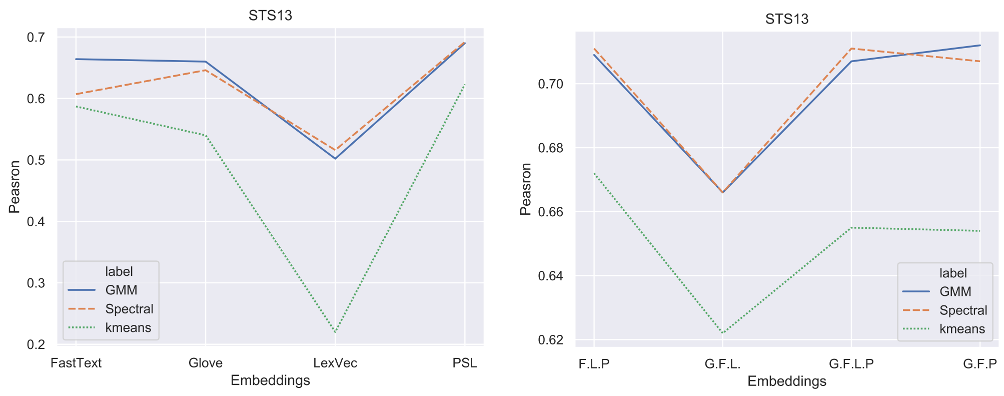
Fair Evaluation. To evaluate and compare our methods with others, we use SentEval Conneau and Kiela (2018), which is a toolkit for evaluating sentence embeddings published by Facebook AI. It is easy to use and contains almost all downstream tasks needed. It shows fairness in evaluating because we only need to rewrite an interface to feed sentence embeddings into the toolkit and others remain the same. It means we compare our method with other models on the same datasets using the same training strategy and the same parameters. The only difference is the sentence embedding.

Experiment Details

Parameter Setting. In the experiment, we test several cluster numbers ranging from 5 to 100. We compared 3 clustering algorithms: Kmeans, GMM, and Spectral Clustering. Rücklé et al. (2018) and Conneau et al. (2017) show that almost all downstream tasks are beneficial from the increased dimensions of word embeddings. We use 4 basic word embeddings and test different concatenations: GloVe Pennington et al. (2014), FastText Bojanowski et al. (2017), LexVec Salle et al. (2016) and PSL Wieting et al. (2015). Each word embedding has 300 dimensions.

Different concatenations are tested on STS13 Agirre et al. (2013) with respect to the Pearson correlations. In Fig 5.1, *F.L.P = FastText + LexVec+PSL*, *G.F.L = GloVe + FastText+LexVec*, *G.F.P = GloVe + FastText + PSL*,*G.F.L.P = GloVe + FastText + LexVec + PSL*. Fig 5.1 shows that Kmeans is inferior to GMM and Spectral Clustering(SC) in all possible concatenations. As for GMM and SC, the performances are close. When the dimensionality comes to 900 and 1200, *G.F.L.P* shows negligible superiority to *G.F.P*. But the increased dimensions also leads to increased computational complexity. In the experiments, we concatenate 3 word embeddings and compare two clustering algorithms (GMM and SC).

The best result is reported.



(a) Dim: 300 (b) Dim: 900 and 1200

Figure 5.1: Different concatenations tested on STS13

We compare our method with both parameterized methods and non-parameterized methods. The evaluated results of other models using SentEval are compared with results shown in Perone et al. (2018) and Vargas et al. (2019) to guarantee correct implementations.

1. Average of GloVe word embeddings;
2. Average of FastText word embeddings;
3. Average embeddings of BERT[1]: the average of all tokens in the sentence from the second layer to the last second layer;
4. BERT [CLS] token from the last layer[2]: The [CLS] token is used in the classification task in BERT;
5. SIF Arora et al. (2016b): a strong baseline using the weighted sum of word embeddings;
6. P-means Rücklé et al. (2018): The monolingual model is used;
7. Semantic subspace sentence embedding (S3E) Wang et al. (2020);
8. SkipThought Kiros et al. (2015): the extended version of skip-gram (SG);
9. Infersent Conneau et al. (2017): the encoder using BiLSTM with Max polling trained on the Stanford Natural Language Inference datasets;
10. Universal Sentence Encoder Cer et al. (2018): the encoder of the transformer trained on the Stanford Natural Language Inference datasets. There are two models available. The transformer encoder is used for comparison;
11. Sentence-BERT (SBERT) Reimers and Gurevych (2019): Siamese BERT-Networks;
12. SBERT-WK Wang and Kuo (2020): the fusion of word representations crossing layers from BERT. It achieves the state-of-the-art performance on STS;

Model	**Dim**	**STS12**	**STS13**	**STS14**	**STS15**	**STS16**	**Ave.**
Non-parameterized methods							
GloVe BOW	300	0.52	0.50	0.54	0.56	0.51	0.53
FastText BOW	300	0.58	0.58	0.65	0.68	0.64	0.63
SIF (GloVe)	300	0.57	0.63	0.68	0.69	0.69	0.65
p-means	3600	0.54	0.52	0.63	0.66	0.67	0.60
S3E (GloVe)	300	0.49	0.64	0.65	0.70	0.67	0.63
Parameterized methods							
BERT BOW	768	0.52	0.50	0.59	0.65	0.58	0.57
BERT [CLS]	768	0.20	0.30	0.20	0.37	0.38	0.29
SkipThought	300	0.41	0.29	0.40	0.46	0.52	0.42
InferSent	4096	0.59	0.58	0.69	0.71	0.71	0.66
USE (Transformer)	512	0.61	0.64	0.71	0.74	**0.74**	0.69
SBERT	768	0.65	0.68	0.73	0.74	0.70	0.7
SBERT-WK†	768	**0.70**	0.68	**0.75**	**0.77**	**0.74**	**0.73**
Proposed Method	900	0.61	**0.71**	0.74	0.76	**0.74**	0.71

Table 5.1: Results of the textual similarity tasks. The Pearson correlations are reported. The best results are shown in bold face.The state-of-the-art model is marked by †.

The results are depicted in Tab 5. Using the [CLS] token from the last layer of BERT has really bad results in capturing the semantic similarities. Wang and Kuo (2020) shows that the representations of tokens in different layers remain stable except for the first layer and the last layer. In the last layer, the token embeddings have a significant change because it learns information associated with the specific training tasks: Masked Language Model and Next Sentence Prediction. The average of the token from the second layer to the last layer shows the equal performance with the average of GloVe. But it does not reveal the full power of BERT. Since the similarities are computed using the cosine distance, it treats each dimension equally Reimers and Gurevych (2019). SBERT-WK Wang and Kuo (2020) Assigns different weights to each dimension and could have surprising performance.

Compared with non-parameterized models, our method improves performance by 10 to 20 points. In parameterized models, SBERT-WK outperforms others by a large margin on STS12. But our method is slightly inferior to SBERT-WK on STS13 to STS16 (less than 1 point). It is still a strong competitor.

Out of Vocabulary (OOV) Words. Our method can easily deal with the words that are not appear in the vocabulary. Since the word is cut into pieces just like what BERT does. If an OOV word comes, it will be cut into pieces first and use the average of centroids associated to pieces that are in the vocabulary.

[1]Bert-base-uncased with 12 layers

[2]the [CLS] from the last layer is not a meaningful representation of the sentence as it is trained for classification.

Computational Speed

In this section, we compare the computational speed among the proposed model, the average of GloVe (GloVe BOW), SBERT Devlin et al. (2018), InferSent Conneau et al. (2017), USE Cer et al. (2018), and SBERT-WK Wang and Kuo (2020). We compute the sentence embedding using *Custrev.pos* dataset [3]. It includes 2412 sentences with various lengths. The average length is 20 words per sentence. GloVe BOW and the proposed method in this paper only rely on Numpy. InferSent, SBERT, and SBERT-WK are based on Pytorch. Universal Sentence Encoder (USE) uses Tensorflow. For USE, we use the *universal-sentence-encoder-large* that deploys on Tensorflow hub[4].

For our method, we start the time counting after the clustering of all word embeddings. Since the clustering is a one-time job. After clustering, the word and its corresponding centroid are stored and no further computation needed. As for USE, InferSent, and SBERT, the time of loading models is ignored. The experiment is conducted on a machine with Intel i7-8650U CPU @1.90GHZ and NVIDIA GeForce GTX 1060, CUDA V10.2. The results are shown in Fig 5.2.

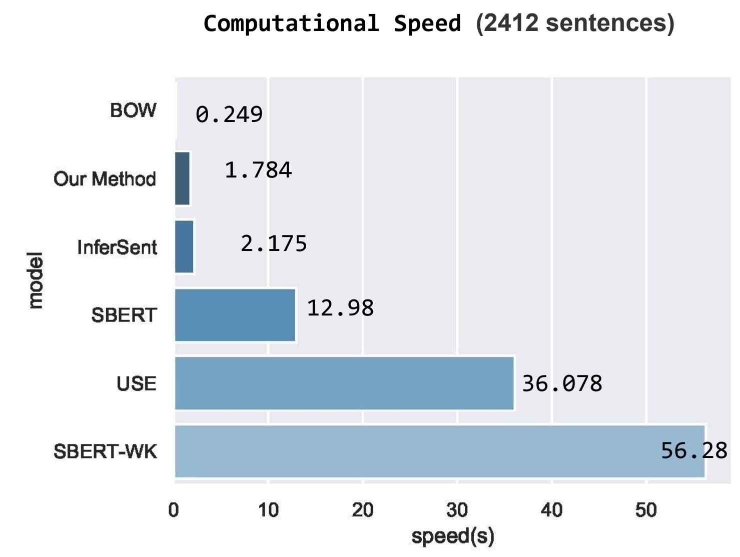


Figure 5.2: The comparison of the computational speed

SBERT-WK has the best performance but with the lowest speed. InferSent is the fastest among parameterized models because of its simple architecture. After clustering, the speed of our methods is still much faster than other complex models. BOW is no doubt the fastest in comparison with others.

[3]https://nlp.stanford.edu/∼sidaw/home/projects:nbsvm
[4]https://tfhub.dev/google/universal-sentence-encoder-large/5

Chapter 6

Conclusions

In this paper, we present a non-parameterized method that maps sentences into the vector space, which relies on the locally-aggregated word embeddings. To choose a good clustering algorithm, we shed light on the reason why clustering algorithms have an impact on the quality of sentence embeddings. We evaluated our method on Semantic Textual Similarity (STS) datasets, where it achieves state-of-the-art performance among non-parameterized methods. It is the second-best method among parameterized and non-parameterized methods.

Our method is computationally efficient. It is about 31 times faster than SBERT-WK, 20 times faster than the Universal Sentence Embedding (USE), and 7 times faster than SBERT. By considering its performance and speed, it is still a worthy method to use when you need to compare the semantic similarities of sentences.

The power of parameterized models is to have general good performances on all downstream tasks. For example, BERT can be easily used on downstream tasks by adding one feed-forward layer on top of it to get surprising results. But when it comes to some specific tasks, compared with the parameterized model, non-parameterized methods have their own advantages. We show that non-parameterized methods have the power to be competitive with parameterized methods on specific tasks. They are much faster and have lower memory requirements.

The proposed method has an implicit assumption: Bag-of-Words. For example, *A likes B* and *B likes A* will have the same sentence embedding in our method. This is one of the problems we discussed in Chapter 2. In the remaining time of the internship, we will improve our method by considering the positional information of words.

Bibliography

Agirre, E., Banea, C., Cardie, C., Cer, D., Diab, M., Gonzalez-Agirre, A., Guo, W., Lopez-Gazpio, I., Maritxalar, M., Mihalcea, R., et al. (2015). Semeval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability. In *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)*, pages 252–263.

Agirre, E., Banea, C., Cardie, C., Cer, D., Diab, M., Gonzalez-Agirre, A., Guo, W., Mihalcea, R., Rigau, G., and Wiebe, J. (2014). Semeval-2014 task 10: Multilingual semantic textual similarity. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 81–91.

Agirre, E., Banea, C., Cer, D., Diab, M., Gonzalez Agirre, A., Mihalcea, R., Rigau Claramunt, G., and Wiebe, J. (2016). Semeval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation. In *SemEval-2016. 10th International Workshop on Semantic Evaluation; 2016 Jun 16-17; San Diego, CA. Stroudsburg (PA): ACL; 2016. p. 497-511.* ACL (Association for Computational Linguistics).

Agirre, E., Cer, D., Diab, M., and Gonzalez-Agirre, A. (2012). Semeval-2012 task 6: A pilot on semantic textual similarity. In ** SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 385–393.

Agirre, E., Cer, D., Diab, M., Gonzalez-Agirre, A., and Guo, W. (2013). * sem 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43.

Arora, S., Li, Y., Liang, Y., Ma, T., and Risteski, A. (2016a). A latent variable model approach to pmi-based word embeddings. *Transactions of the Association for Computational Linguistics*, 4:385–399.

Arora, S., Liang, Y., and Ma, T. (2016b). A simple but tough-to-beat baseline for sentence embeddings.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

Butnaru, A. M. and Ionescu, R. T. (2017). From image to text classification: A novel approach based on clustering word embeddings. *Procedia computer science*, 112:1783–1792.

Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al. (2018). Universal sentence encoder for english. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 169–174.

Conneau, A. and Kiela, D. (2018). Senteval: An evaluation toolkit for universal sentence representations. *arXiv preprint arXiv:1803.05449*.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ethayarajh, K. (2018). Unsupervised random walk sentence embeddings: A strong but simple baseline. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 91–100.

Ganitkevitch, J., Van Durme, B., and Callison-Burch, C. (2013). Ppdb: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764.

Ionescu, R. T. and Butnaru, A. (2019). Vector of locally-aggregated word embeddings (vlawe): A novel document-level representation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 363–369.

Jégou, H., Douze, M., Schmid, C., and Pérez, P. (2010). Aggregating local descriptors into a compact image representation. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 3304–3311. IEEE.

Jegou, H., Perronnin, F., Douze, M., Sánchez, J., Perez, P., and Schmid, C. (2011). Aggregating local image descriptors into compact codes. *IEEE transactions on pattern analysis and machine intelligence*, 34(9):1704–1716.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.

Kusner, M., Sun, Y., Kolkin, N., and Weinberger, K. (2015). From word embeddings to document distances. In *International conference on machine learning*, pages 957–966.

Levy, O. and Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185.

Martinet, J. (2014). From text vocabularies to visual vocabularies what basis? In *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, volume 2, pages 668–675. IEEE.

Mekala, D., Gupta, V., Paranjape, B., and Karnick, H. (2016). Scdv: Sparse composite document vectors using soft clustering over distributional representations. *arXiv preprint arXiv:1612.06778*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Ng, A. Y., Jordan, M. I., and Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856.

Nikolentzos, G., Meladianos, P., Rousseau, F., Stavrakas, Y., and Vazirgiannis, M. (2017). Multivariate gaussian document representation from word embeddings for text categorization. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 450–455.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Perone, C. S., Silveira, R., and Paula, T. S. (2018). Evaluation of sentence embeddings in downstream and linguistic probing tasks. *arXiv preprint arXiv:1806.06259*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Rubner, Y., Tomasi, C., and Guibas, L. J. (1998). A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 59–66. IEEE.

Rücklé, A., Eger, S., Peyrard, M., and Gurevych, I. (2018). Concatenated power mean word embeddings as universal cross-lingual sentence representations. *arXiv preprint arXiv:1803.01400*.

Salle, A., Idiart, M., and Villavicencio, A. (2016). Matrix factorization using window sampling and negative sampling for improved word representations. *arXiv preprint arXiv:1606.00819*.

Torki, M. (2018). A document descriptor using covariance of word vectors. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 527–532.

Vargas, F., Brestnichki, K., and Hammerla, N. (2019). Model comparison for semantic grouping. *arXiv preprint arXiv:1904.13323*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Wang, B., Chen, F., Wang, Y., and Kuo, C.-C. J. (2020). Efficient sentence embedding via semantic subspace analysis. *arXiv preprint arXiv:2002.09620*.

Wang, B. and Kuo, C.-C. J. (2020). Sbert-wk: A sentence embedding method by dissecting bert-based word models. *arXiv preprint arXiv:2002.06652*.

Wieting, J., Bansal, M., Gimpel, K., and Livescu, K. (2015). From paraphrase database to compositional paraphrase model and back. *Transactions of the Association for Computational Linguistics*, 3:345–358.

Wu, L., Yen, I. E., Xu, K., Xu, F., Balakrishnan, A., Chen, P.-Y., Ravikumar, P., and Witbrock, M. J. (2018). Word mover's embedding: From word2vec to document embedding. *arXiv preprint arXiv:1811.01713*.

Yang, Z., Zhu, C., and Chen, W. (2019). Parameter-free sentence embedding via orthogonal basis. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 638–648.

Zipf, G. (1949). Selective studies and the principle of relative frequency in language (cambridge, mass, 1932). *Human Behavior and the Principle of Least-Effort (Cambridge, Mass*.